Introduction

In today's digital age, credit card transactions are a daily occurrence for millions of consumers worldwide. While this provides immense convenience and efficiency, it also opens up substantial opportunities for fraud. Credit card fraud can range from unauthorized use of stolen card information to more complex fraud schemes like synthetic identity fraud. Detecting and preventing these fraudulent activities are crucial, not only for protecting consumers but also for maintaining the integrity of the financial system.

Goal: Use machine learning algorithms to predict credit card fraud, and test the accuracy of it.

Data: We will use Credit Card Fraud Detection data from Kaggle.

Literature

There are many machine learning algorithms to accomplish this task. In the paper Random forest for credit card fraud detection (Xuan, S., et al. 2018), the authors used two kinds of random forests to train the behavior features of normal and abnormal transactions. While in Supervised machine learning algorithms for creditcard fraudulent transaction detection (Mohammed, E., et al. 2018), the authors applied different supervised machine learning algorithms to detect credit card fraudulent transactions. Although some are simple and some are more complex, it is important to choose the right algorithm based on the specific characteristics of the data and the desired outcome of the analysis. In this project, we will examine the accuracy of Random Forest classifiaction then the MSE of XGBoost, on this dataset. The former is a classification algorithm, and the latter is a regression algorithm.

Data

> The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
>
> It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are `'Time'` and `'Amount'`. Feature `'Time'` contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning.
>
> -- Kaggle

Feature **`'Class'`** is the response variable and it takes value 1 in case of fraud and 0 otherwise.

```
import pandas as pd
import numpy as np
df = pd.read_csv("creditcard.csv")
df = df.dropna()
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

Methods

Since the frauds are known and marked in the dataset, it would be recommanded to use supervised learning algorithms. Therefore, Random Foret which combines multiple dcision tress to improve accuracy would be a good choice.

First, we need to look at the summaries of variables.

```
df.describe()
```

```
{"type":"dataframe"}
```

Instance: There are 284807 instances, each represent a credit card transaction, 0.1727% of them are fraudulent.

Target Variable: The target variable is **`Class`**. 1 represents a fraud and 0 otherwise. As in the data description, it is really imbalanced, so a simple accurarcy score may not be sufficient.

Features: We will start will all 30 features.

Seems like the data is already cleaned, so we will go ahead and split the train and test set with `test_size=0.2`.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
roc_curve, auc
import matplotlib.pyplot as plt

X = df[['Time', 'Amount'] + [f'V{i}' for i in range(1, 29)]]
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

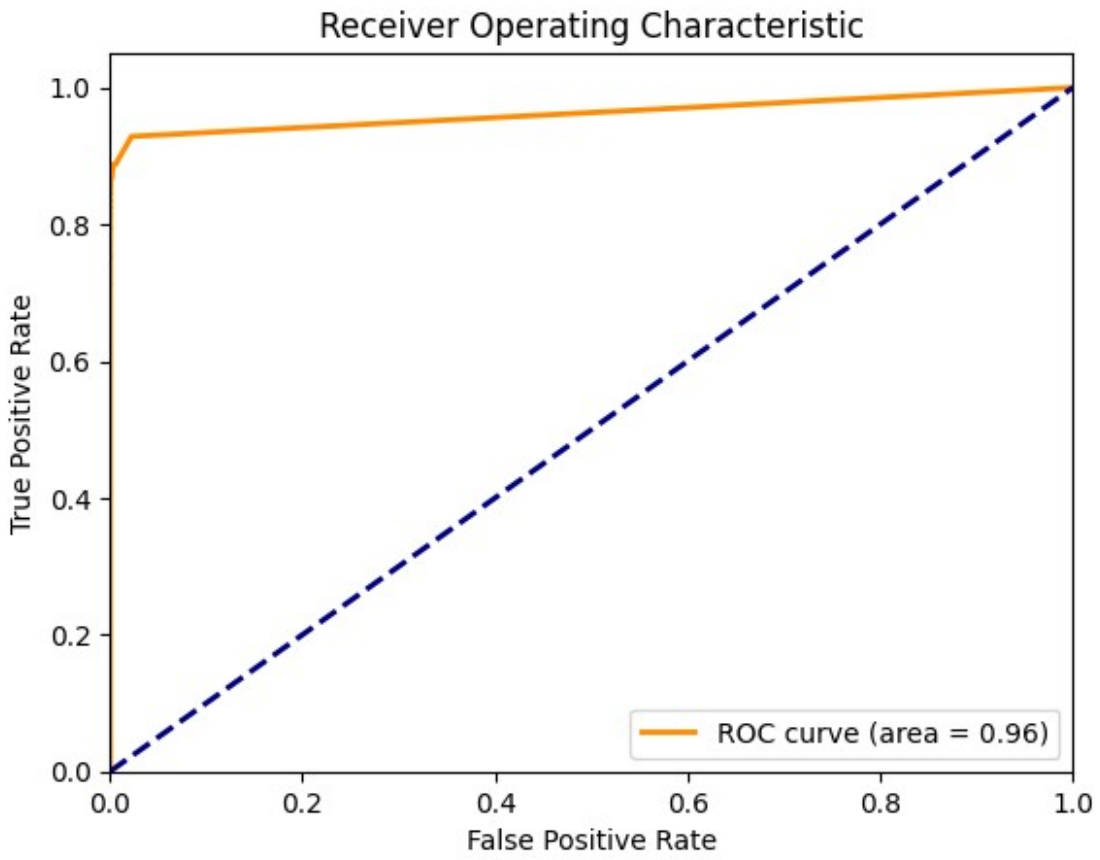
Random Forest

First, we will fit a Random Forest first with `n_estimators=100`, and calculate its accuracy, precision and ROC curve.

```
rf = RandomForestClassifier(n_estimators=100, random_state=42,n_jobs=-
1)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.6f}')
precision = precision_score(y_test, y_pred, average='binary')
print(f'Precision: {precision:.6f}')
```

```
Accuracy: 0.999579
Precision: 0.962500
```

```
y_prob = rf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

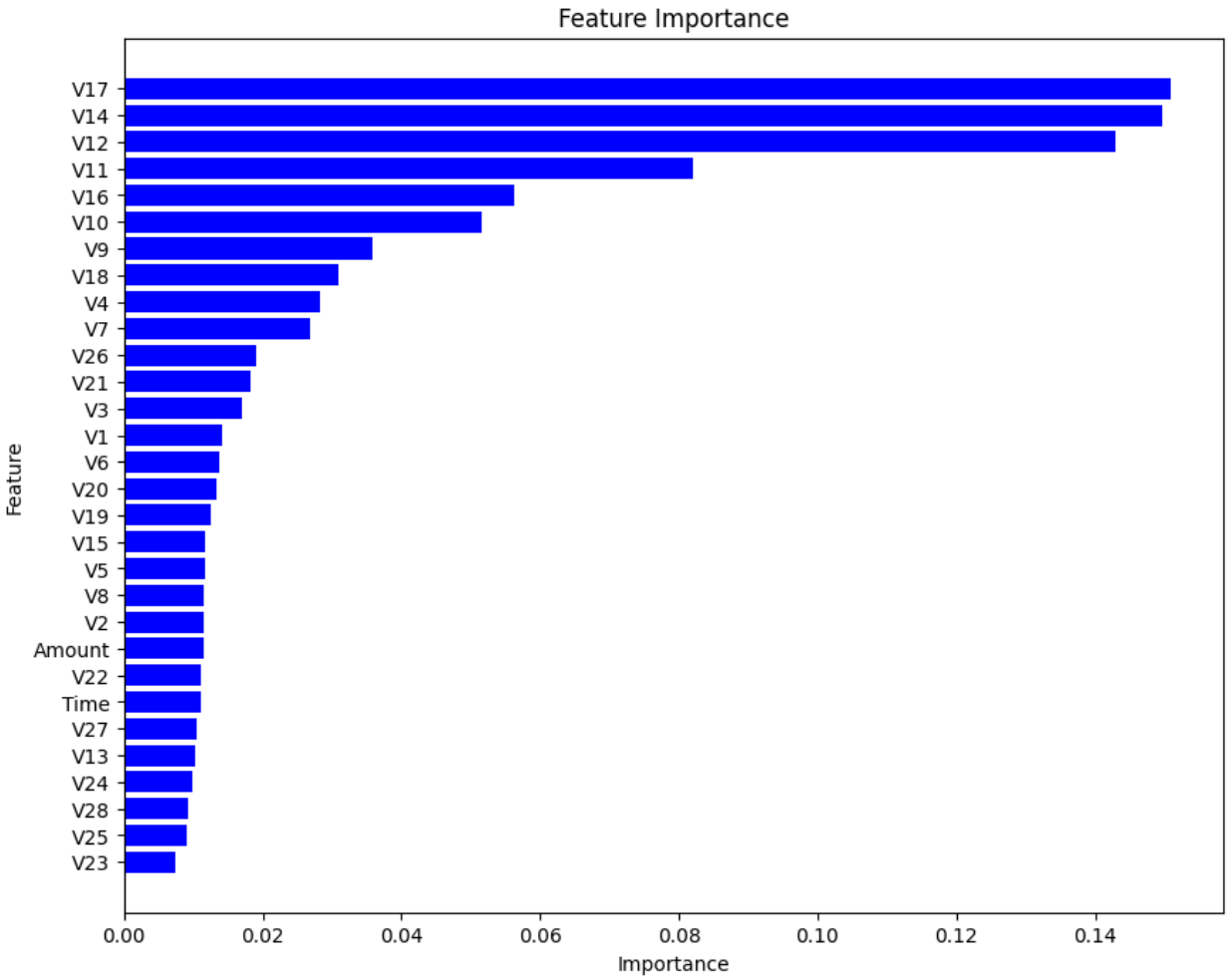


The accuracy and precision are both indeed close to 1. The algorithm is performing exceptionally well in this situation.

The ROC curve quickly rises towards the upper left corner of the plot, which implies that the classifier is able to achieve high true positive rates while maintaining low false positive rates.

However, in this model we used 30 features which introduced the problem of overfitting, so we will use feature selection to address this.

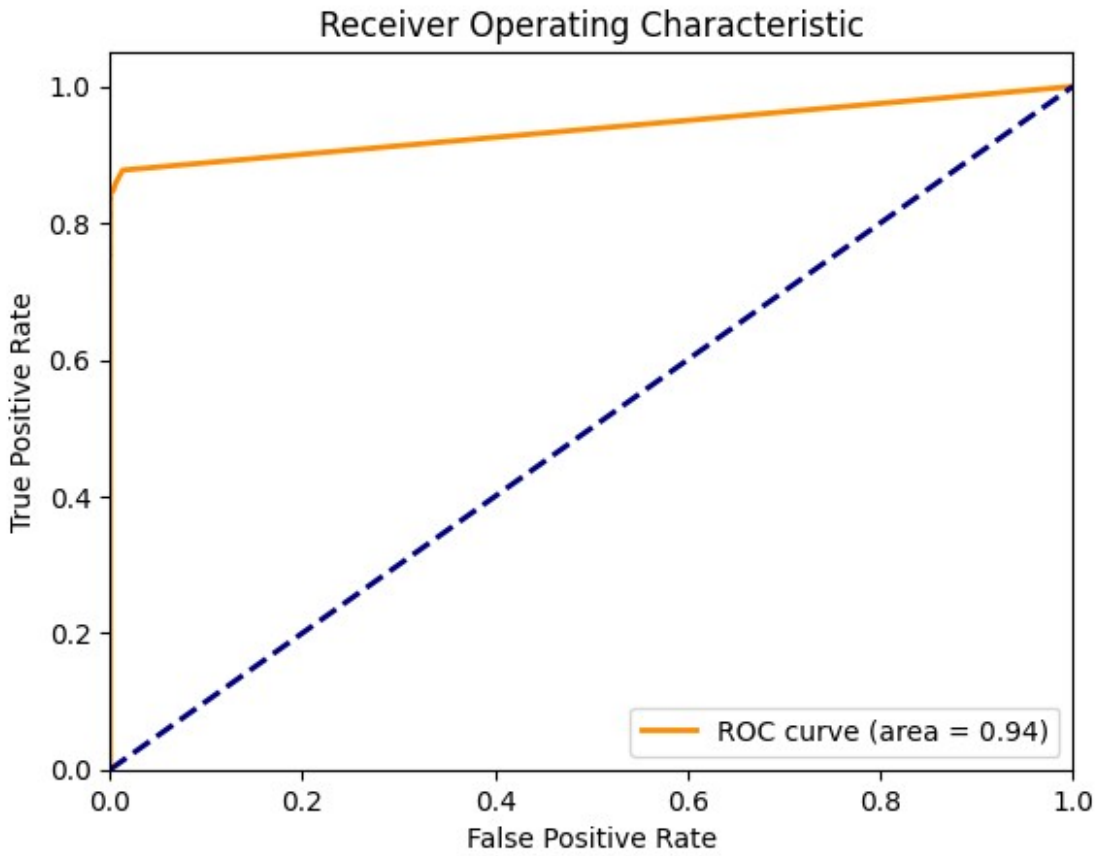
```
feature_importances = rf.feature_importances_
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance':
feature_importances}).sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 8))
plt.barh(importance_df['Feature'], importance_df['Importance'],
color='blue')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.gca().invert_yaxis()
plt.show()
```



The most useful features are `V17`, `V14`, `V12` and `V11`, so we will reduce the features in X to only them.

```
X = df[['V17', 'V14', 'V12', 'V11']]
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
rf = RandomForestClassifier(n_estimators=100, random_state=42,n_jobs=-
1)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.6f}')
precision = precision_score(y_test, y_pred, average='binary')
print(f'Precision: {precision:.6f}')


Accuracy: 0.999526
Precision: 0.949367
```

```
y_prob = rf.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



With only four features, the accuracy only decreased a little bit, the precision decreased to about 0.95 from 0.9625, and the ROC area decrased to 0.94. The model is still performing well and it does take less time to build.

We could also test the overfitting with cross validation.

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
```

```
scores = cross_val_score(rf, X, y, cv=StratifiedKFold(n_splits=3),
scoring='accuracy')
print("Stratified Accuracy: %0.6f (+/- %0.6f)" % (scores.mean(),
scores.std() * 2))
```

```
Stratified Accuracy: 0.999221 (+/- 0.000153)
```

When doing stratified 3-fold cross validation on the model, the accuracy is still over 99%, it did really well.

Now, it is time to do grid search.

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [30, 70, 100],
    'max_depth': [10, 15, 20, None],
}
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=3,
    scoring='accuracy',
    verbose=2,
    n_jobs=-1
)
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
```

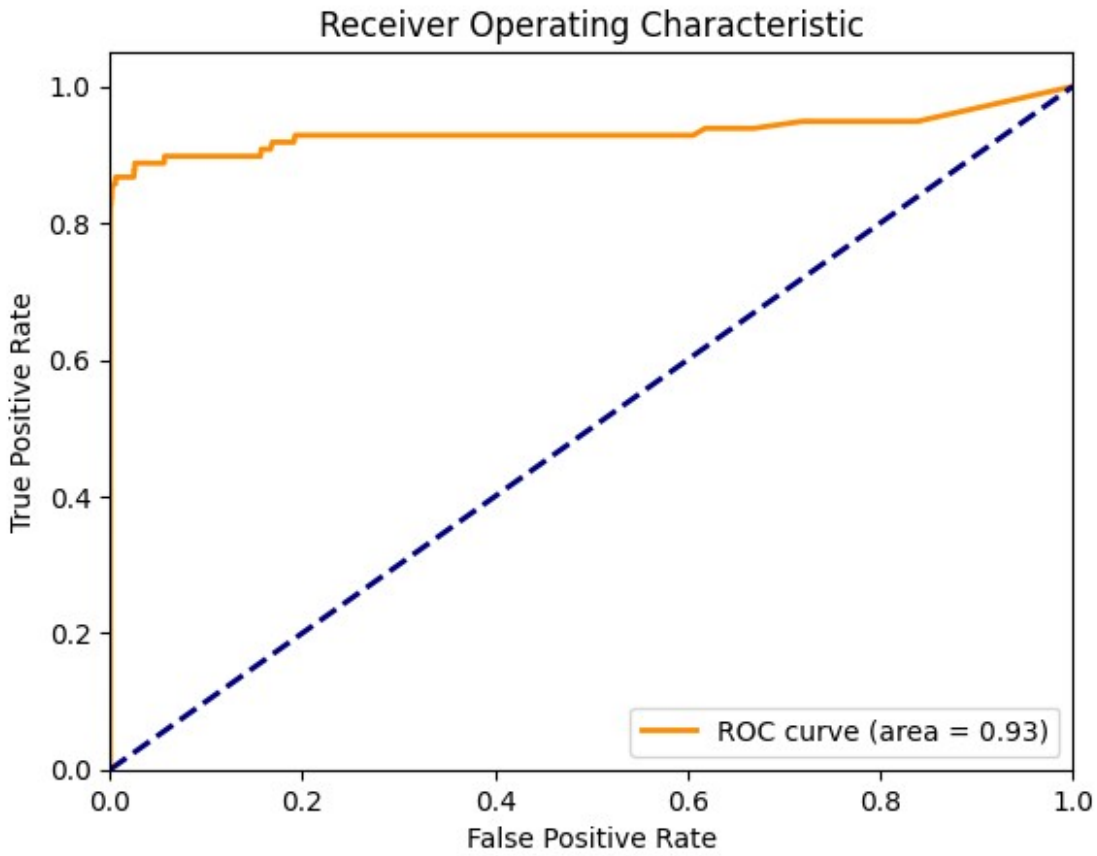
```
Fitting 3 folds for each of 12 candidates, totalling 36 fits

/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/
backend/fork_exec.py:38: RuntimeWarning: os.fork() was called.
os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  pid = os.fork()

Best parameters: {'max_depth': 15, 'n_estimators': 30}
```

The grid search result suggests that `n_estimators=30` and `max_depth=15`, now we will test its accuracy and report its performance.

```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.6f}')
precision = precision_score(y_test, y_pred, average='binary')
print(f'Precision: {precision:.6f}')
```

```
Accuracy: 0.999350
Precision: 0.876543
```

```
y_prob = best_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```



Surprisingly, all the measures are lower indicating that it actually did not perform well compared to the original parameters with `n_estimators=100`. It would be best to put more parameters in, but running this grid search is already taking 10 minutes at least, if adding more parameters or features, it would be impossible.

We are also interested in another algorithm, XGBoost, which is a regression designed to handle a variety of data types and provide robust predictive performance.

Grading Boosting Regression

Now, we could also fit the dataset with a regression algorithm, the XGBoost which processes a little faster.

```
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

X = df[['Time', 'Amount'] + [f'V{i}' for i in range(1, 29)]]
y = df['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
xgb = XGBRegressor(random_state=42)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean Squared Error:', mse)
print('Mean Absolute Error:', mae)
print('R-squared:', r2)
```

```
Mean Squared Error: 0.00044495922817054244
Mean Absolute Error: 0.0013305860648562544
R-squared: 0.7409239927505544
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

xgb = XGBRegressor(random_state=42)
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
}
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=3,
scoring='neg_mean_squared_error', verbose=2, n_jobs=-1)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

print('Best Parameters:', best_params)
y_pred = best_estimator.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Mean Squared Error:', mse)
```

```
print('Mean Absolute Error:', mae)
print('R-squared:', r2)
```

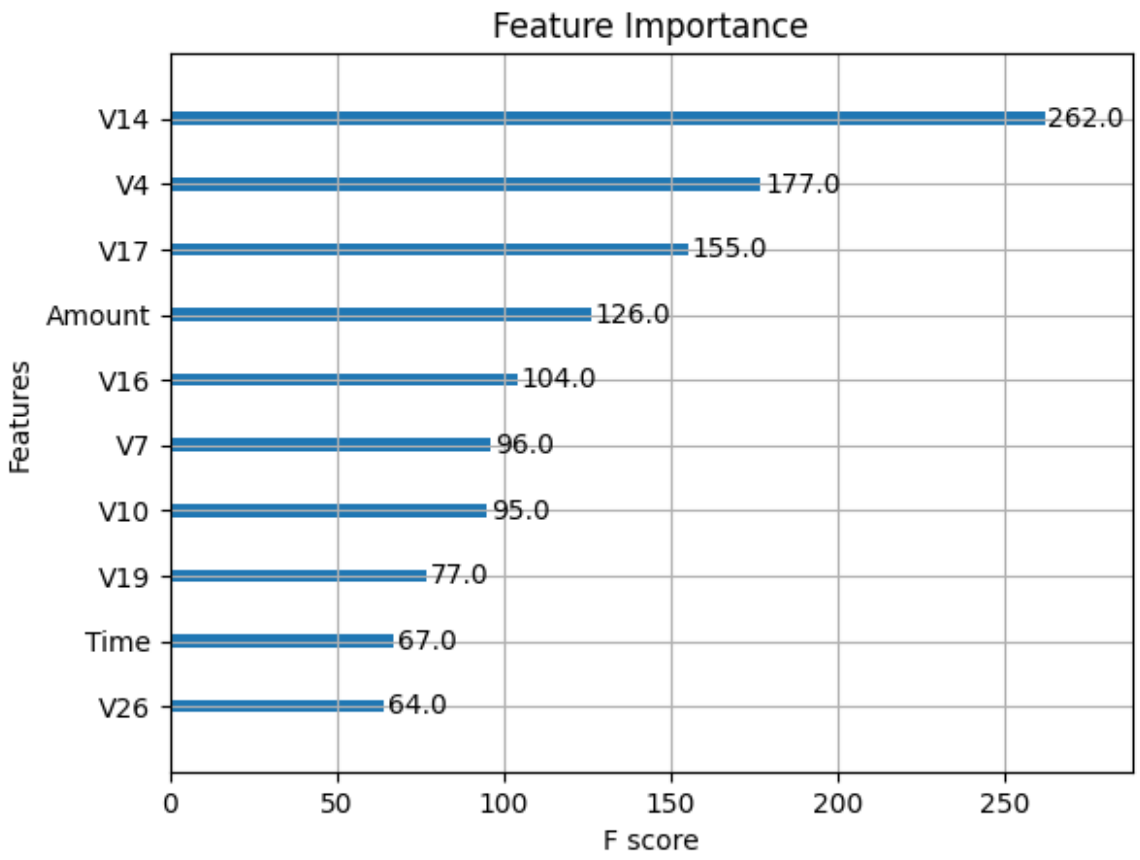
```
Fitting 3 folds for each of 27 candidates, totalling 81 fits

/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/
backend/fork_exec.py:38: RuntimeWarning: os.fork() was called.
os.fork() is incompatible with multithreaded code, and JAX is
multithreaded, so this will likely lead to a deadlock.
  pid = os.fork()

Best Parameters: {'learning_rate': 0.1, 'max_depth': 3,
'n_estimators': 300}
Mean Squared Error: 0.00038773105585047116
Mean Absolute Error: 0.0011920076605091798
R-squared: 0.7742449027310632
```

The result suggests that the `n_estimators=300` for the best MSE, and potentially higher. The MSE and MAE are both extremely low, the predictions are close to the actual values.

We will plot the feature importance to compare it with the Random Forest.

```
from xgboost import plot_importance

plt.figure(figsize=(10, 8))
plot_importance(best_estimator, max_num_features=10)
plt.title('Feature Importance')
plt.show()
```

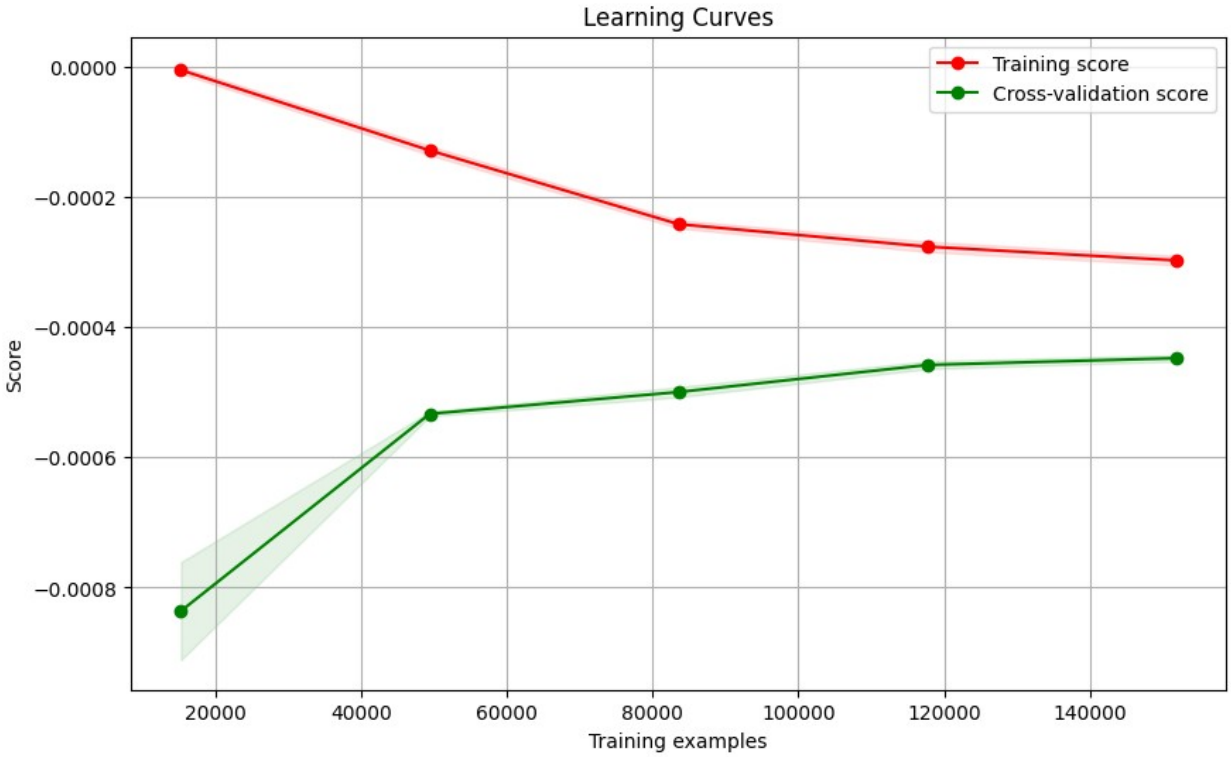
```
<Figure size 1000x800 with 0 Axes>
```



From XGBoost, the most important features are `V14`, `V4`, `V17`, and `Amount`. It appears that `V14` and `V17` are important to both two models.

```
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
import seaborn as sns

best_estimator =
XGBRegressor(n_estimators=300,learning_rate=0.1,max_depth=3,random_sta
te=42)
best_estimator.fit(X_train,y_train)
y_pred = best_estimator.predict(X_test)

def plot_learning_curve(estimator, X, y, cv=None, n_jobs=-1,
train_sizes=np.linspace(0.1, 1.0, 5),
scoring='neg_mean_squared_error'):
    train_sizes, train_scores, test_scores = learning_curve(estimator,
X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring=scoring)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
```

```
    plt.figure(figsize=(10, 6))
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean -
train_scores_std, train_scores_mean + train_scores_std, alpha=0.1,
color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
label="Cross-validation score")

    plt.xlabel('Training examples')
    plt.ylabel('Score')
    plt.legend(loc='best')
    plt.title('Learning Curves')
    plt.show()

plot_learning_curve(best_estimator, X_train, y_train, cv=3)
```



The gap between the training score and the cross-validation score indicates the amount of overfitting. As both curves become closer with more data, it suggests that the model is becoming better at generalizing.

Since the cross-validation score continues to improve and has not plateaued, adding even more data might further enhance the model's performance.

Conclusion

For this imbalanced dataset of credit card fraud detection, both the Random Forest and XGBoost perform exceptionally well, demonstrating robust accuracy, precision, and generalization capabilities. Their ability to handle the complexities of the dataset and effectively manage the class imbalance showcases their suitability for such predictive tasks. Implementing these models with appropriate hyperparameter tuning and evaluation metrics has resulted in reliable and high-performing classifiers that can be crucial in identifying fraudulent transactions.

Bibliography

S. Xuan, G. Liu, Z. Li, L. Zheng, S. Wang and C. Jiang, "Random forest for credit card fraud detection," 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), Zhuhai, China, 2018, pp. 1-6, doi: 10.1109/ICNSC.2018.8361343.

S. Dhankhad, E. Mohammed and B. Far, "Supervised Machine Learning Algorithms for Credit Card Fraudulent Transaction Detection: A Comparative Study," 2018 IEEE International Conference on Information Reuse and Integration (IRI), Salt Lake City, UT, USA, 2018, pp. 122-125, doi: 10.1109/IRI.2018.00025.

V. N. Dornadula and S. Geetha, "Credit Card Fraud Detection using Machine Learning Algorithms," in *Procedia Computer Science*, vol. 165, pp. 631-641, 2019. [Online]. Available: https://doi.org/10.1016/j.procs.2020.01.057

Afriyie, J. K., Tawiah, K., Pels, W. A., Addai-Henne, S., Dwamena, H. A., Owiredu, E. O., Ayeh, S. A., & Eshun, J. (2023). A supervised machine learning algorithm for detecting and predicting fraud in credit card transactions. Decision Analytics Journal, 6, 100163. https://doi.org/10.1016/j.dajour.2023.100163

Kearns, C. (2024, May 3). Credit card fraud detection using machine learning. Cambridge Intelligence. https://cambridge-intelligence.com/detect-credit-card-fraud-with-network-visualization/